# Noise-contrastive estimation of normalising constants and GANs

## Contents

# 1 Fonctions génériques

```
library(ggplot2)
library(reshape)
library(matrixStats)
library(knitr)
```

## 1.1 Algorithme d'Hasting

Utilité : simuler selon $p_m(., \psi)$ pour un paramètre $\psi$ choisi.

| Argument | Type | Exemple | Indication |
|---|---|---|---|
| x | vecteur | rcauchy(100, 0, 1) | notre échantillon de densité inconnue |
| n | entier | 100 | taille de la simulation |
| psi | vecteur | c(0,1) | paramètres de la fonction h |
| h | fonction | | fonction qui retourne $\overline{p_m}(., \psi)$ |

```
hasting = function(x, n, psi, h){
  y = c()
  y = append(y, sample(sample(x, 1)))
  for (i in 2:n){
    y_ = rnorm(1, y[i-1], 1)
    u = runif(1)
    if ( u <=
         (h(y_,psi) * dnorm(y_, y[i-1], 1))
         /(h(y[i-1],psi) * dnorm(y[i-1], y_, 1))
    ){
      y = append(y, y_)}
    else {
      y = append(y, y[i-1])
    }
  }
  return (y)
}
```

Note : on peut très certainement écrire sous forme matricielle cette fonction pour une meilleure performance.

## 1.2 MC MLE

Utilité : retourne une estimation des paramètres selon la méthode décrite dans le papier de Geyer.

```
mc_mle = function(x, psi, h){

  m = length(x)

  y = hasting(x, m, psi, h)

  L = function(theta){
    return(sum(log(h(x,theta)/h(x,psi))) - m*log(mean(h(y,theta)/h(y,psi))))
  }

  theta = optim(
    par = rep(1,length(psi)),
    gr = "CG",
```

```
    control = list(fnscale=-1),
    fn = L
  )$par

  return(theta)
}
```

## 1.3 NCE

Utilité : Retourne l'estimation de la constante et des paramètres.

| Argument | Type | Exemple | Indication |
|---|---|---|---|
| x | vecteur | rcauchy(100, 0, 1) | notre échantillon de densité inconnue |
| law_y | fonction | rnorm | fonction qui retourne un échantillon suivant la loi $p_n$ |
| n | entier | 100 | taille de l'échantillon de bruit suivant la loi $p_n$ |
| params_y | vecteur | c(0,1) | arguments de la fonction law_y |
| log_pm | fonction | | fonction qui retourne le logarithme de la densité $p_m$ |
| log_pn | fonction | | fonction qui retourne le logarithme de la densité $p_n$ |
| size_theta | entier | 3 | taille de $\theta$, vaut habituellement 2 ou 3 |
| method | string | "CG" | méthode d'optimisation, habituellement "CG" ou "BFGS" |

```
nce = function(x, law_y, params_y, log_pm, log_pn, size_theta, n){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)

  h = function(u, theta){
    return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
  }

  J = function(theta){
    return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
  }

  theta = optim(
    par = rep(1, size_theta),
    gr = "CG",
    control = list(fnscale=-1),
    fn = J
  )$par

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

## 1.4 Graphiques

Utilité : afficher l'histogramme pour un échantillon de données $x$.

```
print_hist = function(x) {
  df = data.frame(x = x)
  hist_x = ggplot(df, aes(x=x)) +
           geom_histogram(aes(y = stat(count)/sum(count)), bins = 20, color="white") +
           theme(aspect.ratio = 1) +
```

```
          labs(y = "Fréquence") +
          ggtitle("Distribution de l'échantillon x")
  print(hist_x)
}
```

Utilité : pour NCE, afficher l'évolution des paramètres au fur et à mesure de l'augmentation de n (la dimension de l'échantillon de bruit)

```
NCE_evol_params = function(x, law_y, params_y, log_pm, log_pn, size_theta, ratio, steps, labels) {

  # Creation de l'abscisse
  m = length(x)
  N = seq(0, m*ratio, length.out = steps + 1)

  # Creation de l'ordonnée
  theta = c()
  for (n in N) {
    theta = append(theta, nce(x, law_y, params_y, log_pm, log_pn, size_theta, n))
  }

  # Formatage des données
  theta = t(rbind(matrix(theta, nrow = size_theta),N))
  df = as.data.frame(theta)
  df_melted = melt(df, id.vars = "N")

  # Plot
  plot_df = ggplot(df_melted, aes(x = N, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  geom_point(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par rapport au bruit",
       x = "n (taille du bruit)",
       y = "Paramètres",
       color = "Légende") +
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))

  print(plot_df)

  #return(theta)
}
```

Note : il faudrait optimiser le temps de calcul de ces fonctions, peut-être en matriciel au lieu des boucles ou bien avec du calcul en parralèle sur CPU/GPU

# 2 Applications

## 2.1 Exemple basique : la loi normale

Soit $x$ l'échantillon de taille $m$ obtenu selon la loi de densité inconnue $p_d$.

On considère ici que $p_d$ appartient à la famille de fonctions paramétrées par $\theta = (c, \mu, \sigma)$ suivante :

$$p_m(u;\theta) = \frac{1}{Z(\mu,\sigma)} \times exp\big[-\frac{1}{2}\big(\frac{u-\mu}{\sigma}\big)^2\big] \quad \text{d'où} \quad ln(p_m(u;\theta)) = c - \frac{1}{2}\big(\frac{u}{\sigma} - \frac{\mu}{\sigma}\big)^2$$

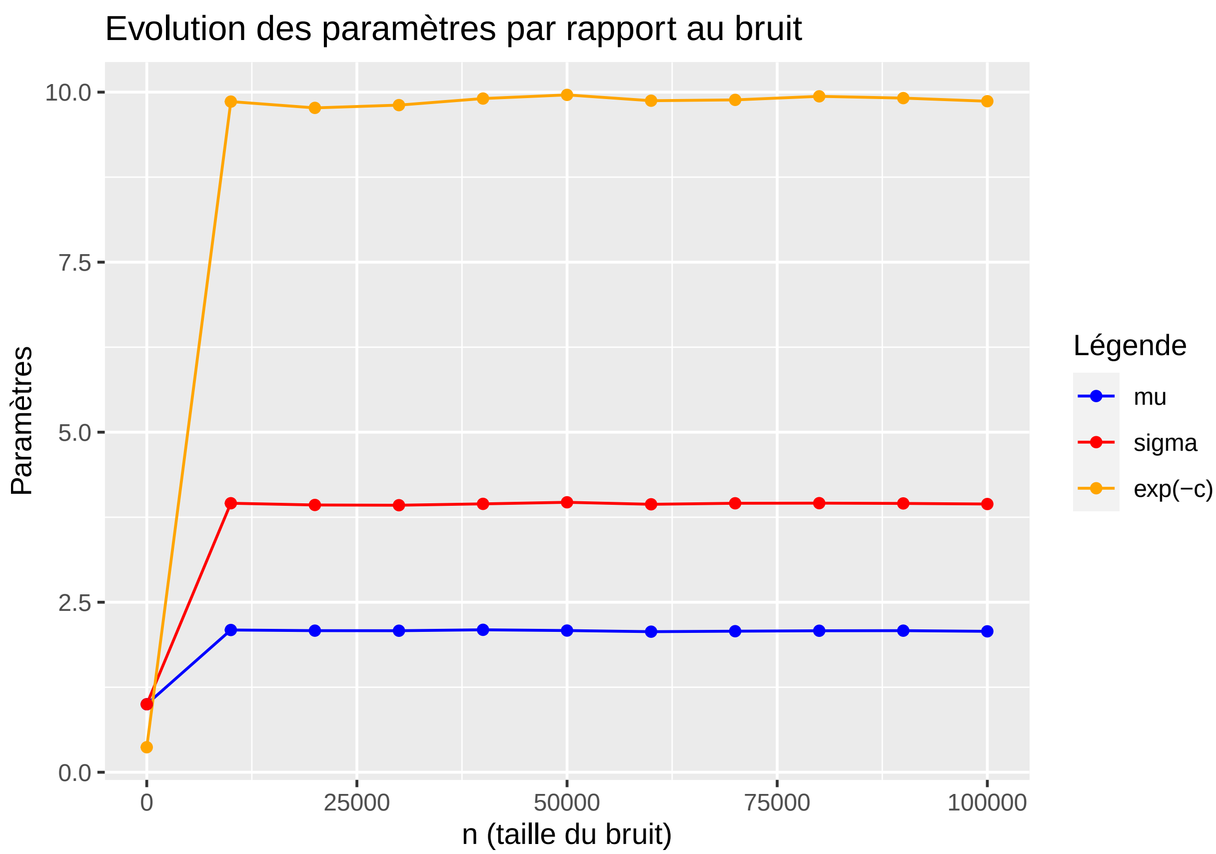
```
pm_barre = function(u, theta){
  return(exp(-0.5 * ((u - theta[1]) / theta[2]) ** 2))
}

log_pm = function(u,theta){
  return(theta[3] - 1/2 * (u/theta[2] - theta[1]/theta[2]) ** 2)
  # theta[1] = mu / theta[2] = sigma / theta[3] = c
}

log_pn_cauchy = function(u){
  return(log(dcauchy(u, mean(x), sd(x))))
}

m = 10000
n = 10000
x = rnorm(m, 2, 4)
size_theta = 3

# METHODE MC MLE
mc_mle(x, c(mean(x),sd(x)), pm_barre)
```

```
## [1] 1.548471 4.034881
```

```
# METHODE NCE
nce(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n)
```

```
## [1] 2.031212 3.960471 9.903040
```

```
NCE_evol_params(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, 10, 10, c("mu", "sigma
```


Evolution des paramètres par rapport au bruit
10.0
7.5
5.0
2.5
0.0
Paramètres
0
25000
50000
75000
100000
n (taille du bruit)
Légende
mu
sigma
exp(−c)

# 3 Nouvelles approches

## 3.1 Bootstrap

```r
# Calcul de {size_boot} estimateurs par bootstrap
NCE_bootstrap = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, size_boot, labels) {
  m = length(x)
  theta_bootstrap = c()
  x_bootstrap = x
  for (i in 1:size_boot) {
    theta_bootstrap = append(theta_bootstrap, nce(x_bootstrap,
                                              law_y,
                                              params_y,
                                              log_pm,
                                              log_pn,
                                              size_theta,
                                              n))
    x_bootstrap = sample(x, size = m, replace=TRUE)
  }
  return(matrix(theta_bootstrap, nrow = size_theta))
}

# Plot la moyenne empirique des estimateurs bootstrap en fonction du nombre d'estimateurs
NCE_bootstrap_plot = function(matrix_theta_bootstrap) {

  # Formatage des données pour plot
  array_boot = 1:length(matrix_theta_bootstrap[1,])
  df = as.data.frame(cbind(t(rowCumsums(matrix_theta_bootstrap))/array_boot,array_boot))
  df_melted = melt(df, id.vars = "array_boot")

  # Plot
  plot_df = ggplot(df_melted, aes(x = array_boot, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par bootstrap",
       x = "Taille du bootstrap",
       y = "Paramètres",
       color = "Légende") +
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))
  print(plot_df)
}


# etude bootstrap de l'estimateur
bootstrap = function(matrix, alpha){
  return(data.frame(
    theta = matrix_theta_bootstrap[,1],
    biais = rowMeans(matrix_theta_bootstrap) - matrix_theta_bootstrap[,1],
    IC = rowQuantiles(matrix_theta_bootstrap, probs = c(alpha/2, 1-alpha/2))
  ))
}
```

```r
x_test = rnorm(1000,2,4)

matrix_theta_bootstrap = NCE_bootstrap(x_test, rcauchy, c(mean(x_test),sd(x_test)), log_pm, log_pn_cauc
```

```r
kable(bootstrap(matrix_theta_bootstrap, 0.05))
```

| theta | biais | IC.2.5. | IC.97.5. |
|---:|---:|---:|---:|
| 2.159328 | -0.0995386 | 1.695646 | 2.420927 |
| 3.930428 | 0.0461366 | 3.758227 | 4.189884 |
| 9.718872 | 0.2281341 | 9.218138 | 10.656809 |

### 3.2 Récursivité

Utilité : améliorer récursivement la précision de l'estimation via les estimations précédentes

```r
mc_mle_recursif = function(x, psi, h, size_of_loop){

  m = length(x)

  for (i in 1:size_of_loop) {
    y = hasting(x, m, psi, h)

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - m*log(mean(h(y,theta)/h(y,psi))))
    }

    psi = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    print(psi)
  }

  return(psi)
}
```

```r
mc_mle_recursif(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 2.133248 3.969706
## [1] 1.817023 3.939416
## [1] 1.477700 4.026385
## [1] 2.823350 4.037071
## [1] 2.303140 3.942716
## [1] 2.002994 3.776283
## [1] 2.272293 3.615115
## [1] 1.829352 4.017812
## [1] 2.382251 3.766056
## [1] 3.129564 4.166704
```

```
## [1] 3.129564 4.166704
```

```r
mc_mle_recursif_2 = function(x, psi, h, size_of_loop){

  m = length(x)
  M = m
```

```
  for (i in 1:size_of_loop) {
    y = hasting(x, M, psi, h)

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - M*log(mean(h(y,theta)/h(y,psi))))
    }

    theta = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    x = append(x, hasting(x, m, theta, h))
    M = M + m

    print(theta)
  }

  return(theta)
}
```

```
mc_mle_recursif_2(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 2.003932 4.409048
## [1] 2.313613 4.129167
## [1] 1.964361 4.044733
## [1] 2.379613 4.052961
## [1] 2.394489 4.067971
## [1] 2.606421 3.945720
## [1] 2.375067 3.913493
## [1] 2.348457 3.999414
## [1] 2.367065 4.037187
## [1] 2.156409 3.964303
```

```
## [1] 2.156409 3.964303
```

```
mc_mle_recursif_3 = function(x, psi, h, size_of_loop){

  m = length(x)
  M = m
  y = hasting(x, m, psi, h)

  for (i in 1:size_of_loop) {

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - M*log(mean(h(y,theta)/h(y,psi))))
    }

    theta = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
```

```
      fn = L
    )$par

    x = append(x, hasting(x, m, theta, h))
    y = append(y, hasting(x, m, psi, h))
    M = M + m

    print(theta)
  }

  return(theta)
}
```

```
mc_mle_recursif_3(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 2.182595 3.985400
## [1] 2.326984 3.817137
## [1] 2.326246 3.721515
## [1] 2.297959 3.719952
## [1] 2.245231 3.775098
## [1] 2.099776 3.788388
## [1] 2.064852 3.786430
## [1] 2.051490 3.794675
## [1] 2.031545 3.816378
## [1] 2.055703 3.826313
```

```
## [1] 2.055703 3.826313
```

```
nce_recursif = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, size_of_loop){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)

  for (i in 1:size_of_loop){

    h = function(u, theta){
      return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
    }

    J = function(theta){
      return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
    }

    theta = optim(
      par = rep(1, size_theta),
      gr = "CG",
      control = list(fnscale=-1),
      fn = J
    )$par

    print(theta)

    y = do.call(law_y, c(list(n),theta[-size_theta]))
```

```r
  }

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

```r
nce_recursif(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n, 10)
```

```
## [1]  2.018243  3.980133 -2.309206
## [1]  2.150177  3.975920 -2.295197
## [1]  2.066614  3.941924 -2.293835
## [1]  2.078741  3.950962 -2.306488
## [1]  2.071498  3.956436 -2.296585
## [1]  2.105809  3.967361 -2.294061
## [1]  2.100096  3.912653 -2.284646
## [1]  2.067672  3.979017 -2.308197
## [1]  2.124943  3.929724 -2.280958
## [1]  2.057197  3.991671 -2.312538
```

```
## [1]  2.057197  3.991671 10.100022
```